# **Pontificia Universidad Católica Madre y Maestra**

# **Facultad de Ciencias de la Ingeniería**

# **Departamento de Ingeniería en Sistemas y Computación**

# Logo PUCMM (Color).png

# **Programación Web 1**

# **ST-ISC-415-T-001**

# **Práctica #4:**

# “Creación de Blog utilizando ORM - JPA”

# **Estudiantes:**

Daniel Pérez, 2011-0839

Ariel Salcé, 2009-0707

# **Profesor:**

Ing. Carlos A. Camacho G.

# **Fecha de entrega:**

# Lunes, 13 de Junio del 2016

Santiago de los Caballeros, República Dominicana

Introducción

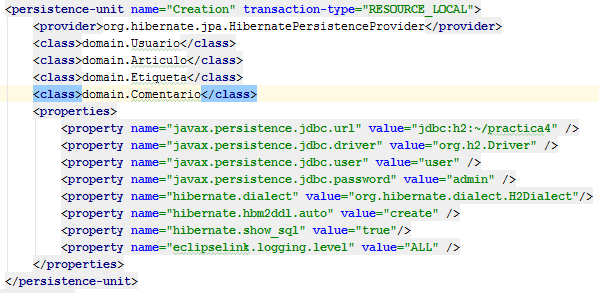
Esta semana, se exige por el profesor realizar las mismas especificaciones de la práctica #3, con la diferencia de que en esta ocasión se utilizará ORM para que nuestras entidades de objetos sean directamente mapeadas a las entidades en el ambiente relacional de la base de datos (en este caso se seguirá con H2 como motor de base de datos). Se utilizará Hibernate para la persistencia de los datos. Además de esto, se agregaron 3 nuevas funcionalidades que no se encuentran en la práctica #3 y estas son como se presentan a continuación:

* En la página de inicio debemos implementar un sistema de paginación que muestre únicamente 5 publicaciones por página, las demás publicaciones serán cargadas en función a la página correspondiente que estemos seleccionando, en función a la cantidad existente de artículos.
* Implementar la opción de seleccionar una etiqueta y visualizar todos los artículos, asociados a dicha etiqueta.
* Incluir un conteo de valoración positivo (likes) y negativo (dislikes) de artículos y comentarios. Solo los usuarios registrados sin importar el rol, pueden realizar valoraciones.

Desarrollo

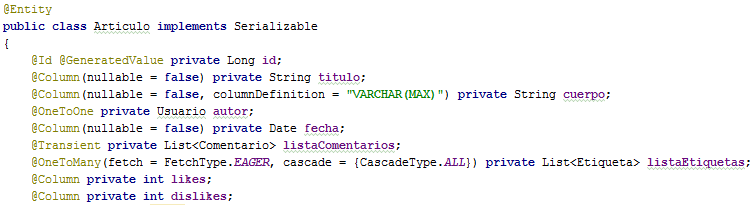
Lo primero que se realizó fue establecer la asociación entre las entidades POJO con el modelo relacional, es decir, que las clases que se tiene en el modelo de objetos se conviertan en tablas para el modelo relacional. Para lograr esto, se deben seguir una serie de pasos que son como se denota a continuación:

**Configurar JPA**: Para poder configurar JPA, se debe trabajar con el archivo de configuración persistence.xml que debe estar ubicado en la carpeta META-INF de la carpeta resources. Es aquí donde se coloca la configuración necesaria para poder realizar la conexión a la base de datos.



Como podemos ver en la imagen, se establece el proveedor de la persistencia, se declaran las entidades que van a formar parte de la persistencia y luego se establecen las configuraciones para la conexión a la base de datos.

**Asociación**: Para realizar esta asociación, arriba de las clases POJO se coloca la anotación @Entity, con la cual se indica que esa clase forma parte de la persistencia. En cada clase, como en los modelos relacionales, existen (valga la redundancia) relaciones entre los objetos y estas se especifican también con anotaciones en nuestras clases, donde estas pueden ser: @OneToOne, @OneToMany, @ManyToOne o @ManyToMany. También, como en todos los modelos relacionales, las tablas deben poseer alguna llave primaria como un identificador (ID) de esas tablas y estos IDs se especificaron desde el ámbito de objetos con la anotación @Id, que indica que el atributo que está a continuación de esa anotación va a ser el identificador de esa tabla. Si se quiere se puede generar automáticamente el ID utilizando la anotación @GeneratedValue luego de la anotación de @Id, indicando que ese ID va a tener un valor autogenerado.

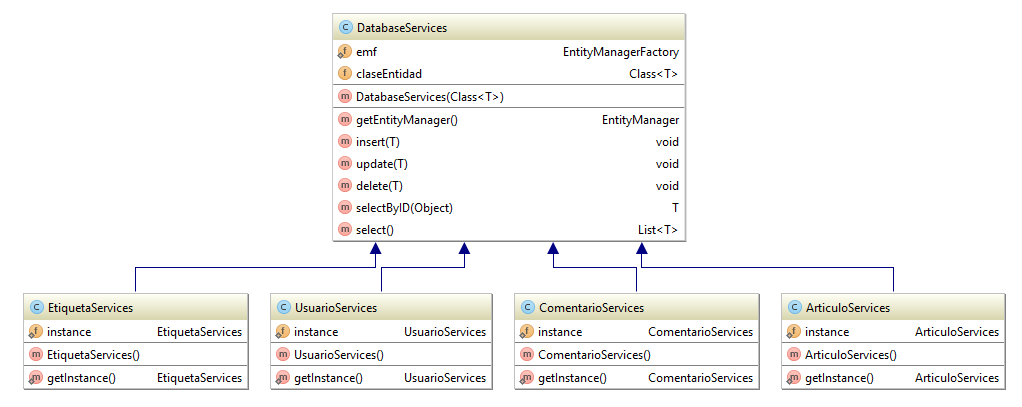


Como se puede ver en la imagen arriba, Artículo forma parte de la persistencia con el @Entity, se expresa que un Articulo tiene muchas etiquetas con la anotación @OneToMany, al igual que el ID que en este caso es autogenerado por Hibernate con la anotacion @Id @GeneratedValue.

También se puede ver que si se quiere con el @Column se crea una columna simple en la tabla y el @Transient para atributos que no van a formar parte de la persistencia.

Siguiendo estas mismas reglas de sintaxis se establecieron todas las otras entidades que se utilizaron en la práctica #3 (Etiqueta, Usuario y Comentario).

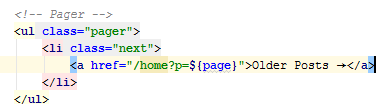
Después de que se establecieron las entidades de manera persistente, el siguiente paso es adaptar la lógica que se tiene de la práctica #3 con la lógica para realizar operaciones en la base de datos por parte de hibernate. Se utilizó al igual que en la práctica #3 DatabaseServices como clase madre, que posee los métodos básicos: insert, update, delete, selectByID y select.



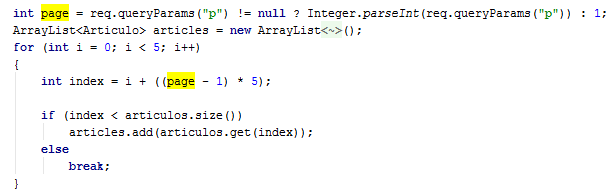
A diferencia de los métodos con los mismos nombres de la práctica anterior, estos utilizan conceptos de persistencia de hibernate. Algunos de los métodos para el manejo de la persistencia son el “***persist****”* para insertar en la base de datos, el “***merge****”* para actualizar y el “***remove***” para eliminar. Además, también a diferencia de la práctica anterior, se manejan transacciones para cada diferente funcionalidad en su método correspondiente, ya sea insertar, eliminar, etc.

Habiendo terminado con lo referente a base de datos, la siguiente fase de la práctica se centra en las nuevas funcionalidades pedidas por el profesor.

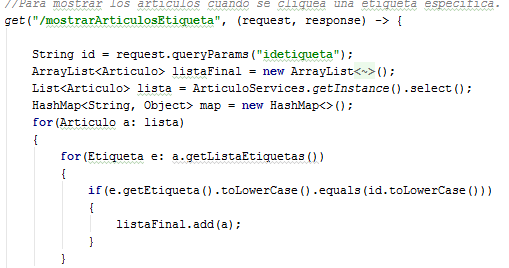
**Paginación**: Para esta se trabajó en la página home que es la principal de la aplicación. Aqui en el FTL se le manda por parametros el número de la pagina para ser procesado en el GET de home.



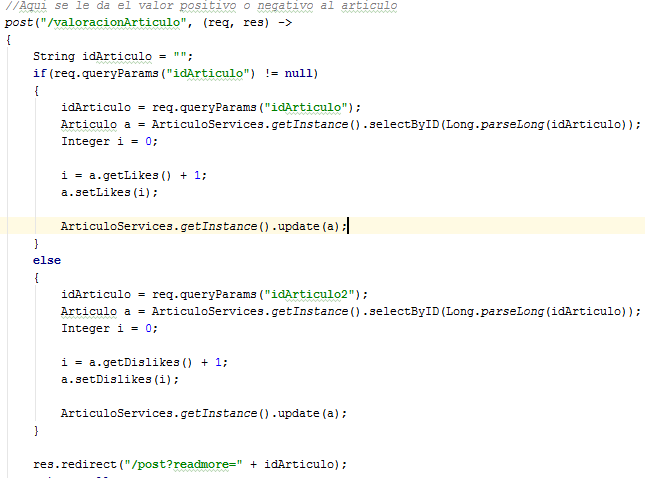
En el GET de home, entonces se revisa si le esta llegando algun numero de pagina actualmente, si no se hace el número actual de la página sea 1 por defecto y si existe el número, se obtiene su valor. Con este valor luego se asegura que por cada pagina máximo sean 5 artículos que se le muestren a los usuarios. En la siguiente imagen se muestra como se trabaja con esto dentro del GET de home:



**Visualizar articulos por etiqueta**: Este se trabaja en una página FTL con lógica parecida al home, donde esta muestra los artículos, pero las etiquetas que son botones al hacerles click activan el respectivo GET en el main y mandan el id la etiqueta cliqueada. Con este id simplemente se revisa cada artículo en existencia y se pregunta cuál de esos tiene alguna etiqueta con ese id específico, entonces esos artículos serán los que se muestran a los usuarios al finalizar este proceso.



**Valoración de Articulos y Comentarios**: Para este se trabajó con 2 GET, donde uno manejaba los Artículos y el otro manejaba los Comentarios. El FTL que se utilizó fue post.ftl, ya que este poseía el artículo completo y los comentarios de dicho artículo, que son los objetos afectados por esta funcionalidad. En el FTL se colocaron dos botones para likes y dislikes en el artículo y en los comentarios, donde ambos botones mandan el id del Artículo o del Comentario dependiendo donde se encuentren. Con este id, simplemente se busca ese Comentario específico en la base de datos o ese Articulo especifico y se actualiza la cantidad de likes o dislikes que tiene dependiendo del botón que se cliqueo (si fue el de like o dislike).



Aquí podemos ver, por ejemplo el GET para los like y dislike de los Artículos, cómo se manejan individualmente entre o likes o dislikes para actualizar los datos.

Para revisar más a fondo lo hecho en esta practica, favor revisar el siguiente repositorio:

<https://github.com/danieldmw2/ProgramacionWeb1>

Conclusión

Aunque ORM fue un poco complicado al principio y la tarea más dificil de la práctica fue posiblemente adaptar la práctica anterior a trabajar con ORM, realmente es un concepto con el que no estábamos muy familiarizados y fue muy interesante el poder crear tablas en la base de datos solamente con el modelo de objetos que se trabajó.

Con respecto a las tareas nuevas, a pesar de que fueron sencillas de realizar, luego nos dimos cuenta que las soluciones a estas no fueron quizás las más óptimas, ya que por ejemplo para los tres se estaba obteniendo la base de datos completa, lo cual si se tienen muchos datos en la base de datos es bastante ineficiente, pero quizás con un query utilizando un count se pudo haber ahorrado el tener que bajar la base de datos completa. Es algo que se va a tratar de tener en cuenta siempre que sea posible.